Extending Feature Models with Relative Cardinalities

Gustavo Sousa, Walter Rudametkin, Laurence Duchien

Extending Feature Models with Relative Cardinalities

Gustavo Sousa
University of Lille, France
CRIStAL UMR 9189
École Centrale de Lille
Inria
gustavo.sousa@inria.fr

Walter Rudametkin
University of Lille, France
CRIStAL UMR 9189
École Centrale de Lille
Inria
walter.rudametkin@inria.fr

Laurence Duchien
University of Lille, France
CRIStAL UMR 9189
École Centrale de Lille
Inria
laurence.duchien@inria.fr

ABSTRACT

Feature modeling is widely used to capture and manage commonalities and variabilities in software product lines. Cardinality-based feature models are used when variability applies not only to the selection or exclusion of features but also to the number of times a feature can be included in a product. Feature cardinalities are usually considered to apply in either a local or global scope. However, we have identified that these interpretations are insufficient to capture the variability of cloud environments. In this paper, we redefine cardinality-based feature models to allow multiple relative cardinalities between features and we discuss the effects of relative cardinalities on feature modeling semantics, consistency and cross-tree constraints. To evaluate our approach we conducted an analysis of relative cardinalities in four cloud computing providers. In addition, we developed tools for reasoning on feature models with relative cardinalities and performed experiments to verify the performance and scalability of the approach. The results from our study indicate that extending feature models with relative cardinalities is feasible and improves variability modeling, particularly in the case of cloud environments.

CCS Concepts

•**Software and its engineering** → **Software product lines;**

Keywords

Feature Model, Cardinality, Constraints

1. INTRODUCTION

Feature modeling is a widely used approach to capture commonalities and variability across software systems that are part of a product line or system family [16]. A feature model is usually depicted as a tree diagram whose nodes represent features that can be selected to build a software product. The tree hierarchy describes a composition relationship between features, while additional constraints refine these relationships.

Several extensions to feature modeling have been proposed since its inception, usually motivated from pragmatic needs in product line engineering. Among these, feature cardinalities were introduced to deal with scenarios where a feature can be selected multiple times for a given product, each time with a possibly different set of subfeatures [4].

The semantics of cardinalities in feature models and its effects on cross-tree constraints have been thoroughly studied and formalized in different ways [6, 19, 21, 25]. Feature cardinalities are interpreted to apply either locally (in relation to its immediate parent feature) or globally (concerning the whole product configuration). However, through our investigation in managing variability in cloud computing platforms, we found that feature cardinalities may also be related to an ascendant feature that is not the direct parent feature.

To deal with this limitation, we introduce the concept of *relative cardinality*, which is a generalization of the existing interpretations of feature cardinalities. In this paper we redefine cardinality-based feature models to take into account relative cardinalities. We then analyze the effects on feature model semantics, including cross-tree constraints and cardinality consistency. Finally, we evaluate the use of this extended feature model definition for modeling variability in cloud computing platforms as well as the scalability of automatically generating and validating configurations.

In Section 2 we identify the limitations we found in feature cardinalities while designing feature models for cloud computing platforms. Section 3 explains the concept of relative cardinality and discusses how it affects cardinality consistency and cross-tree constraints. Section 4 describes how we implemented relative cardinalities into a tool for automatic analysis of feature models and Section 5 evaluates the approach. Finally, we discuss related work in Section 6 and the conclusions in Section 7.

2. MOTIVATION

In the cloud computing paradigm, computing resources are usually delivered to customers in the form of *infrastructure*, *platform* or *software* services. Each cloud computing provider offers a different set of services, at different abstraction levels, such as processing power, network communication, virtual machines, containers, software packages, application servers, databases, development and management tools, etc. To choose a provider, configure a suitable cloud environment, and deploy a cloud application, stakeholders need to be aware of the services available and of all the constraints between them.

Commonalities and variabilities in the providers' services can be captured as feature models, making the selection of a provider suitable for automated processing using a software product line approach. This approach has been employed in recent work to support the automatic selection and configuration of cloud providers [22, 8, 10]. However, in previous work, variability is only considered in

SPLC '16 September 19–23, 2016, Beijing, China

DOI:

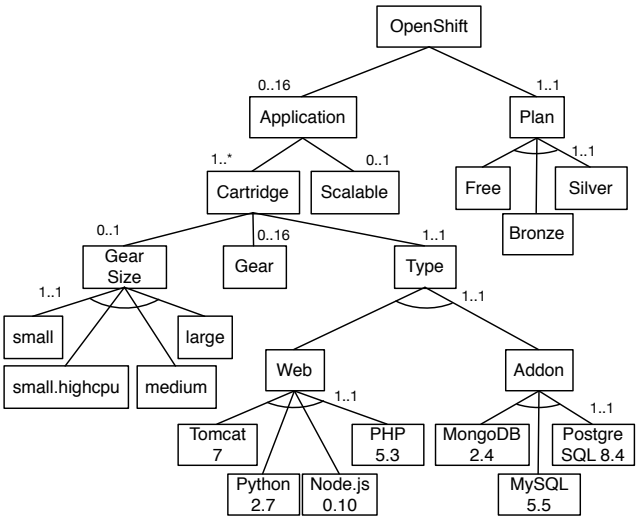


Figure 1: Excerpt from the OpenShift cloud feature model

the context of a resource (e.g. virtual machine, storage, network) or for the deployment of a single application.

When variability is only considered in a specific context, constraints at the provider level are ignored, which leads to invalid configurations. However, while considering variability in a cloud provider's context we identified that existing feature modeling constructs were insufficient to capture all the variability. In the following section we present an example of these limitations and the challenges to overcome them.

2.1 Motivating example

Figure 1 provides an extract of a feature model we designed to capture variability in the OpenShift PaaS provider. In OpenShift, an `Application` is composed of a set of `Cartridges`, which are software features such as application servers, databases, caching services, management tools, etc. `Cartridges` are run by processing nodes called `Gears`. For each `Cartridge` the user can choose the number of `Gears`, as well as the `Gear Size`, which defines the memory and processing capabilities.

OpenShift imposes no limit on the number of `Applications` or `Cartridges` allowed. However, the number of `Gears` is limited by the user's plan. Considering that the limit for a plan is 16 `Gears`, we can have any combination ranging from 1 `Application` with 16 `Gears`, to 16 `Applications` with 1 `Gear` each. To enable users to describe both of these valid configurations, the feature model was designed in a way that the feature cardinalities of `Application` and `Gear` allow for up to 16 instances. Though this modeling allows for expressing all possible valid configurations, it does not prevent users from describing invalid ones. For instance, we could define an application with 16 `Applications`, each one with 16 `Gears`, thus exceeding the provider's limits.

A similar problem can be found when we consider the case of `Web` cartridges. An `Application` can have multiple cartridges, but the provider requires that every `Application` should have exactly one of type `Web`. Nevertheless, the feature model does not provide any information about this restriction, allowing for `Applications` with any number of `Web` cartridges.

This problem occurs because the features `Gear` and `Web` are directly associated to the `Cartridge` feature, but the number of allowed instances are respectively associated to the whole product configuration and to `Application` instances. This means that the number of allowed instances of a given feature may not only be related to its direct parent, but also to ascendant features in the hierarchy or to the whole product configuration.

2.2 Challenges

Providing a way to specify cardinalities in multiple levels would make it possible to capture the variability found in cloud computing configurations and to increase the expressive power of feature models. However, introducing new constructs to feature modeling may affect its semantics.

In this paper we investigate how feature models can be extended to deal with multiple interpretations of the cardinality scope. We then analyze how this extension affects the semantics and the analysis of cardinality-based feature models. More specifically, we seek to deal with the following challenges:

- *Capture relative cardinalities in feature models*. How to extend cardinality-based feature models to consider features for which cardinality interpretation is neither global nor local? That is, to take into account cardinalities that can be associated to features at different levels of hierarchy.
- *Ensure the consistency between relative cardinalities*. What are the criteria for relative cardinalities to be consistent and how do we ensure consistency in such a feature model?
- *Update additional constraints to handle multiple relative cardinalities*. How does the introduction of relative cardinalities affect additional constraints and how can additional constraints be evolved to take relative cardinalities into account?

3. RELATIVE FEATURE CARDINALITIES

Feature cardinalities define the number of instances of a given feature, and their semantics have been previously studied and formalized [6, 19, 25]. Cardinalities are applied either globally or locally. In the first case, all instances of a feature (throughout an entire configuration) are counted, while in the second case, the number of instances is counted for each instance of the parent feature.

Figure 2 shows a feature model with three example configurations that use different interpretations for cardinality $1..2$ of feature E. (a) shows the feature model. In (b) cardinalities are interpreted globally, thus no more instances of E can be added to the configuration (i.e., the limit is 2 in the configuration). In (c) cardinalities are interpreted locally, meaning each instance of D can have 1 or 2 children instances of E.

However, as we described in the OpenShift example in Section 2.1, there are cases where the cardinality context is neither global nor local, but relative to some other feature. This can be seen in Figure 2 (d), where the cardinality $1..2$ of E is, for this example, interpreted as relative to feature B. This means that each instance of B can have 1 or 2 instances of E in its subtree, regardless of the number of instances of D.

3.1 Formalization

We introduce and formalize the concept of *relative cardinality*.

Definition 1. **(Relative cardinality)** The relative cardinality between two features x and y, such that x is descendant of y in the feature diagram, is the interval that defines the minimum and maximum number of x instances for each y instance.

The concept of relative cardinality can be seen as a generalization of the possible interpretations for cardinalities in feature models. In this sense, the cardinality of a feature in relation to the feature model's root is equivalent to its global cardinality. Similarly,

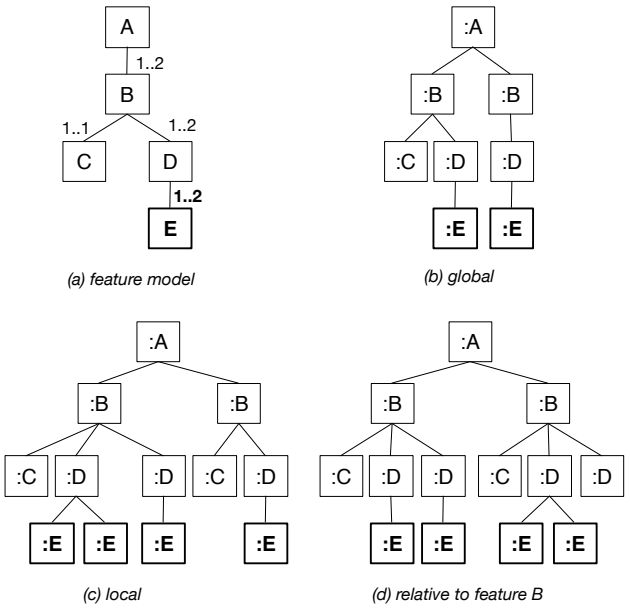


Figure 2: Different interpretations for cardinalities.

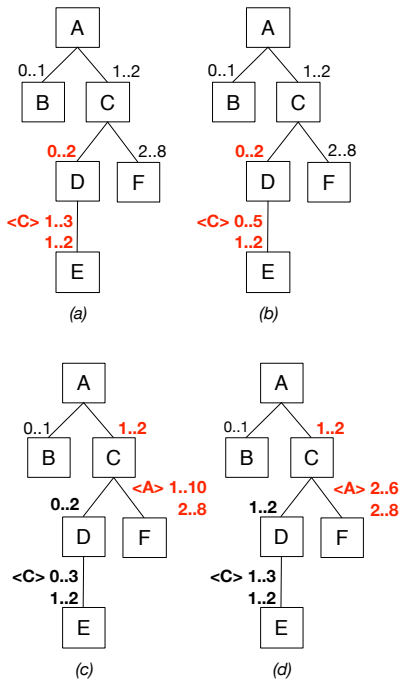


Figure 3: Relative cardinalities consistency

the cardinality of a feature in relation to its parent corresponds to its local cardinality.

Most cardinality-based feature modeling approaches consider that cardinalities apply locally [7, 6, 9, 18, 19, 28]. In this case, the feature diagram hierarchy defines an *implicit relative cardinality* between any feature and its ancestors. As an example, in Figure 2 (a), if we consider cardinalities apply locally we can infer that the implicit relative cardinality between E and B is $1..4$. Each instance of B can have from 1 to 2 instances of D and each instance of D 1 or 2 instances of E. Thus there can be 1 to 4 Es for each B.

Although feature cardinalities can be interpreted locally in many cases, there are scenarios where a local interpretation of cardinalities is insufficient. In these cases there is a mismatch between the implicit and the actual relative cardinality. To deal with this problem, we have extended the existing cardinality-based feature model constructs to consider relative cardinalities as part of their definition. Based on the work done by *Michel et al.* [19], we redefine a feature model as follows:

Definition 2. **(Feature model)** A feature model is a 7-tuple $\mathcal{M} = (\mathcal{F}, G, r, E, \omega, \lambda, \phi)$ such that:

- $\mathcal{F}$ is a non-empty set of features;
- $r \in \mathcal{F}$ is the root feature;
- $E : \mathcal{F} \setminus \{r\} \rightarrow \mathcal{F}$ is a function that represents the *parent-of* relation between features such that its *transitive closure* on $\mathcal{F}$, denoted by E^+, is *irreflexive* and *asymmetric*. These conditions guarantee the tree structure of the feature diagram;
- $\mathcal{C} = \{(x, y) \in \mathbb{N}^2 : x \leq y\}$ is the set of cardinality intervals;
- $\omega : E^+ \rightarrow \mathcal{C}$ is a function that represents the relative cardinality between two features that are part of the *ancestors-of* relation, denoted by E^+, the *transitive closure* of E on $\mathcal{F}$;
- $G \subset \mathcal{F}$ is a possibly empty subset of feature groups;
- $\lambda : G \rightarrow \mathcal{C}$ is a function that represents the group cardinalities of feature groups;
- ϕ is a set of cross-tree constraints (Section 3.3).

The proposed definition updates the cardinality function ω to consider not only one but two features that are related in the feature model hierarchy. By doing so, it gives first-class status to relative cardinalities, which were only implicit in feature models with local cardinalities. This concept is more general and allows for expressing cardinalities at different levels, including global and local cardinalities.

3.2 Cardinalities consistency

By their nature, relative cardinalities allow defining multiple cardinalities for a feature, where each cardinality is relative to a different ancestor feature. In order to be valid, these cardinalities need to be consistent among themselves. Cardinality consistency is linked to the notion of *range consistency* [20]. A cardinality is considered range consistent if each value in its range is used in at least one valid product configuration.

In Figure 3 we show some feature models that were defined with inconsistencies in relative cardinalities. In the given examples, relative cardinalities are described above the feature node using the notation <*X*> $m..n$, where $m..n$ is its cardinality relative to feature X. When no specifier is given, the cardinality is considered to apply locally, in relation to the feature's direct parent.

In the feature model in Figure 3 (a), feature E has, in addition to its local cardinality $1..2$, a cardinality of $1..3$ relative to its ancestor C. However, this cardinality is inconsistent with the local cardinality $0..2$ of feature D. If an instance of C has no instances of D, it is not possible to have instances of E, thus the cardinality $1..3$ between E and C does not hold. In this case there would be no valid products with 0 instances of D, even though the cardinality of D is $0..2$.

In Figure 3 (b), the cardinality of E relative to C allows for up to 5 instances. However, each instance of C allows for a maximum of 2 instances of D, and for each of them a maximum of 2 instances of

E, allowing for a total of 4 instances of E for each instance of C. Thus, the local cardinalities of D and E are not consistent with the $0..5$ relative cardinality between E and C. Sample fixes for these examples are shown in (c) and (d) where either the relative cardinality between E and C, or local cardinality of D, are updated.

Figure 3 (c) shows another example of inconsistency where the relative cardinality between F and A is $1..10$ but from the local cardinalities of C and F we can infer that we should have at least 2 instances of F in any product. Also, in (d) we have an example where the relative cardinality between F and A has a maximum of 6 instances, which conflicts with the $2..8$ local cardinality of feature F, since no valid products can exist with 7 or 8 instances of F.

When cardinalities are inconsistent the number of allowed instances is ambiguous and subject to diverse interpretations. In addition, they do not represent the actual number of allowed feature instances. Based on the notion of *range consistency* and analysis of the semantic relation between multiple relative cardinalities, we define the criteria for cardinality consistency as below.

Definition 3. **(Cardinality consistency)** Given functions min and max that return the minimum and maximum values for a cardinality range, a feature model $\mathcal{M} = (\mathcal{F}, G, r, E, \omega, \lambda, \phi)$ is consistent concerning its relative cardinalities if $\forall x, y, z \in \mathcal{F} \mid (z, y) \in E^+ \ \wedge \ (y, x) \in E^+$ the following conditions hold:

$$min(\omega(z,x)) \geq min(\omega(z,y)) \ \cdot \ min(\omega(y,x)) \tag{1}$$

If each x instance has at least $min(\omega(y,x))$ instances of y and each y instance have at least $min(\omega(z,y))$ instances of z, then the minimum number of z instances for each x instance is at least $min(\omega(z,y)) \ \cdot \ min(\omega(y,x))$.

$$max(\omega(z,x)) \leq max(\omega(z,y)) \ \cdot \ max(\omega(y,x)) \tag{2}$$

If each x instance has at most $max(\omega(y,x))$ instances of y and each y has at most $max(\omega(z,y))$ instances of z, then each x instance can have at most $max(\omega(z,y)) \ \cdot \ max(\omega(y,x))$ instances of z.

$$min(\omega(z,x)) \leq max(\omega(z,y)) \ \cdot \ min(\omega(y,x)) \tag{3}$$

The feature model should enable specifying at least one valid product in which the number of y instances for an x instance is the minimum $min(\omega(y,x))$. In this case, the maximum number of instances of z for each x is $min(y,x) \ \cdot \ max(z,y)$. Thus, if $min(\omega(z,x)) \geq max(\omega(z,y)) \ \cdot \ min(\omega(y,x))$ there would be no valid products using the lower bound of the cardinality $\omega(y,x)$ and the cardinalities would not be consistent.

$$max(\omega(z,x)) \geq max(\omega(z,y)) \ \cdot \ min(\omega(y,x)) \tag{4}$$

Similarly, in at least one product, the number of y instances for each x should be $max(\omega(y,x))$. In this case, the minimum number of instances of z for each x instance is $max(\omega(y,x)) \cdot min(\omega(z,y))$. Thus, if $max(\omega(z,x)) \leq max(\omega(y,x)) \ \cdot \ min(\omega(z,y))$ there would be no product with the maximum cardinality $max(\omega(y,x))$.

$$\begin{aligned} min(\omega(z,x)) \leq \\ min(\omega(z,y)) \ + \ max(\omega(z,y)) \ \cdot \ (max(\omega(y,x)) - 1) \end{aligned} \tag{5}$$

At least one instance of y should have the minimum number of z instances $min(\omega(z,y))$. In this case, if one y instance has this minimum number of z instances, the maximum number of z instances for the x ascendant instance would be reached if all other y instances under the same x had the maximum number of z instances. That said, the maximum number of z instances for this x instance would be $min(\omega(z,y)) \ + \ max(\omega(z,y)) \ \cdot \ (max(\omega(y,x)) - 1)$. Therefore, if the minimum relative cardinality between z and x was greater than this value, there would be no cases where the lower bound of the cardinality $\omega(z,y)$ would be valid.

$$\begin{aligned} max(\omega(z,y)) \geq \\ max(\omega(z,y)) \ + \ min(\omega(z,y)) \ \cdot \ (min(\omega(y,x)) - 1) \end{aligned} \tag{6}$$

At least one instance of y should have the maximum number of z instances $max(\omega(z,y))$. Also, if at least one y has this maximum number, then the ascendant x instance would have at least $max(\omega(z,y)) \ + \ (min(\omega(y,x)) - 1)$. Therefore, if the maximum bounds of $\omega(z,x)$ is less than this value there would be no valid configuration that uses the maximum cardinality of $max(\omega(z,y))$.

This definition establishes that any three features that are linked in an ancestors-descendant relationship should meet the six identified constraints. Together, these constraints guarantee that for any value in the concerning cardinality ranges, at least one valid product can be defined in which this value is employed. Since these conditions should apply for all possible relative cardinality relationships, it guarantees that the feature model is consistent regarding relative cardinalities.

3.3 Multiple cardinalities and constraints

In cardinality-based feature models, additional cross-tree constraints such as *requires* and *excludes* need to be adapted to consider instances of features [19]. *Quinton et al.* [21] extended cross-tree constraints with cardinalities allowing for *requires* constraints over the global or local number of instances of a feature. In [17], authors propose cross-tree constraints that can be applied to individual instances of features. These previous developments allow for defining constraints that can be interpreted in global or local scope, but do not consider relative cardinalities neither how constraints involving multiple scopes can be evaluated.

When we consider multiple relatives cardinalities, additional constraints need to be defined not only over feature cardinalities but also over their relative ones. To enable the definition of additional constraints over relative cardinalities, we update existing constraint notations and define their semantics. We first establish the notion of a *constraining expression*, and based on it, we define the concept of *relative cardinality constraints*.

Definition 4. **(Constraining expression)** A constraining expression is described by a tuple $\varepsilon = (r, c)$ where

- $r \in \mathcal{C}$ is a cardinality range;
- $c \in E^+$ is a pair of features part of the child-ancestor relation in the associated feature model.

We use the notation $[m..n]\ (x,y)$ to describe a constraining expression where $r = (m,n)$ and $c = (x,y)$. Such expressions represent a predicate over the number of instances of feature x that are descendant of an instance of y. Therefore constraining expressions represent a condition over a relative cardinality in a feature model.

Definition 5. **(Relative cardinality constraint)** A constraint is defined as an implication <*C*> $\varepsilon_1 \ \delta \ ... \ \delta \ \varepsilon_n \rightarrow \varepsilon_{cons}$ where:

- $C \in \mathcal{F}$ is a feature that defines the context where the constraint will be evaluated;

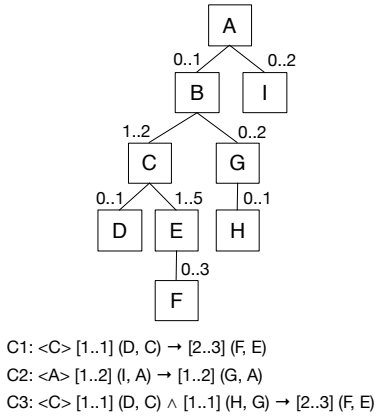


Figure 4: Additional constraints and relative cardinalities

- $\varepsilon_1 ... \varepsilon_n$ and ε_{cons} are constraining expressions;
- $\delta \in \{\wedge, \vee\}$ is a logical operation.

A relative cardinality constraint defines an implication that if the composed left hand constraining expressions $\varepsilon_1\ \delta\ ...\ \delta\ \varepsilon_n$ hold, the right hand ε_{cons} expression should also hold.

As shown in the above definition, constraining expressions are the base elements for describing a constraint. However, though they express a condition over a relative cardinality, they do not express in which context it should be evaluated. For example, considering the feature model in Figure 4, we can define a constraining expression such as $[5..10]\ (F, C)$. Does this expression evaluate to true if one instance of C has 5 to 10 instances of F or if all instances of C have 5 to 10 instances of F?

The role of the context feature in the constraint is to remove this ambiguity and specify in which context expressions should be evaluated. Thus, in constraint C_1 (Figure 4), the <C> context indicates that its constraining expressions should be evaluated individually for each instance of C. For instance, the C_1 constraint expresses that *C instances that have exactly one D instance as a child should have 2 or 3 instances of F for each E instance*. Actually, this constraint redefines the local cardinality of F (relative to its parent E) for some set of C instances, those that have exactly one D instance as a child.

Likewise, in C_2 the <A> context indicates that the constraint has to be evaluated globally, for the singleton instance of the root feature A. In this case, C_2 expresses a global implication that if at least one I instance is part of the product configuration then at least one G instance should also be included.

In C_1 and C_2, all constraining expressions deal with features that are under the subtree of the context features <C> and <A> respectively. The semantics of constraint evaluation are simpler as it suffices to consider each instance of C or A individually (and its subtree) and verify if the constraining expressions hold. However, how can we evaluate a constraint such as C_3, which contains combined expressions at different levels of hierarchy, including outside the context feature?

In this case, expressions whose features are not in the subtree of the context feature are evaluated in the context of the lowest common feature. This is the lowest common ancestor, in the feature diagram tree, of all features involved in the constraint; which in the case of C_3 is feature B.

With the described semantics, C_3 expresses that for each c instance of C and b instance of B such that b is an ancestor of c in a configuration tree, if c has exactly one child instance of D, and if all instances of G that are children of b have exactly one instance of H, then all instances of F that are children of this same c should have 2 or 3 instances of E.

The proposed constructs enable describing both simple and complex constraints involving relative cardinalities. The introduction of context features along with the semantics described above clarify how constraints can be evaluated even when they involve cardinalities from different levels in the feature diagram tree.

3.4 Contribution summary

In our approach for feature modeling, we tackle the challenges identified in Section 2.2 in the following way:

- To enable multiple interpretations of cardinalities, we extended the definition of cardinality-based feature models, replacing feature cardinalities by relative cardinalities.
- To ensure consistency between relative cardinalities, we identified a set of constraint conditions to verify range consistency in the presence of multiple relative cardinalities.
- To consider relative cardinalities in cross-tree constraints, we introduced constraining expressions over relative cardinalities and precise semantics for identifying the constraint's context of evaluation.

4. MODELING AND AUTOMATION

This section discusses the tooling support we developed to enable modeling feature models with relative cardinalities, including language support, inference and consistency analysis, and configuration conformance.

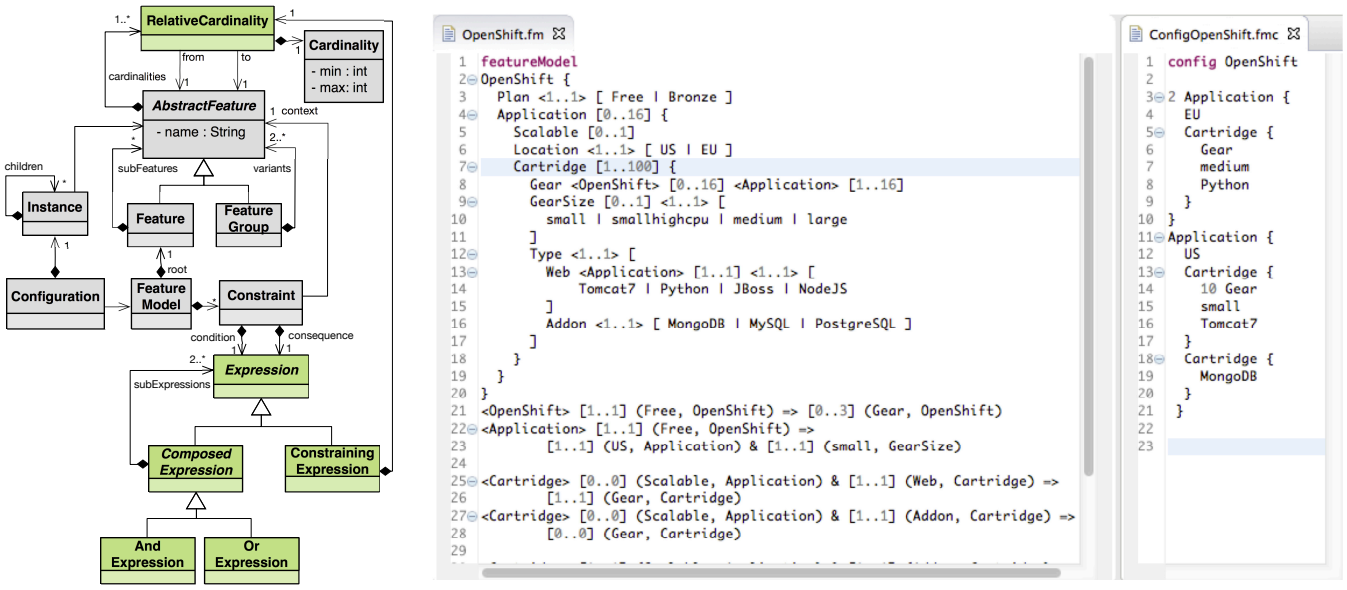
4.1 Modeling

To enable the description of feature models with relative cardinalities, we designed a domain specific language based on the definitions given in Section 3.1. Figure 5 (a) depicts the abstract syntax of the designed language as a metamodel. The classes in gray represent the concepts commonly found in cardinality-based feature models; those in green represent the concepts introduced by our approach. Our language includes the `RelativeCardinality` concept, which is composed by a cardinality range and is associated to two features (`from` and `to`). In addition, elements `Constraint` and `ConstrainingExpression` allow the definition of relative cardinality constraints as described in Section 3.3.

We implemented the feature modeling language on the XTEXT framework [3]. We defined the abstract syntax using the EMF ECORE metamodel [27] and the concrete syntax as a XTEXT grammar. Figures 5 (a) and (b) show the feature model and configuration editors with the motivating feature model described in Section 2.1. The editor plugin, together with the metamodel and grammar are available in the accompanying site[1].

A feature model is described as a tree structure where braces ({}) define the decomposition hierarchy and brackets ([]) feature groups. Feature cardinalities are defined between brackets ([]) and feature groups' cardinalities between less and greater than symbols (<>). Additional relative cardinalities are described using the $<X>\ [m..n]$ notation where $m..n$ is the cardinality relative to feature X. In line 8 of Figure 5 (b) we can see that relative cardinality between `Gear` and `Application` is 1..16. Once a feature model is defined in the feature model DSL, we can check for the consistency of cardinalities and generate configurations for it.

[1]http://researchers.lille.inria.fr/~sousa/relativecard/



(a) metamodel *(b) feature model editor* *(c) configuration editor*

Figure 5: Metamodel and editor for feature models with relative cardinalities.

Configurations are also defined in a hierarchy that matches that of the feature model. The configuration editor allows for describing full or partial configurations. In a partial configuration, not all required feature instances need to be included and intermediary features may be omitted. To avoid repeating identical instances users can also add the number of desired instances of a feature. In Figure 5 (c), lines 3-10 include two `Application` instances with the same decomposition of features. Line 14 states that the third `Application` should have a `Cartridge` with 10 instances of `Gear`. Overall, this example describes a configuration for the OpenShift provider to deploy two Python application services in Europe as well as a Java application service and a Mongo database in the United States. From a feature model configuration we can check if it conforms to the feature model and if it is partial to generate a complete configuration.

4.2 Inference and consistency checking

As we can see from the abstract syntax, the language allows the specification of multiple relative cardinalities. However, in most cases, relative cardinalities match exactly with the implicit relative cardinality derived from local cardinalities. In these cases, the product line designer may not want to describe all the relative cardinalities but rather those that are different from the implicit ones, letting the system infer the remaining cardinalities.

For example, in Figure 5 (b), we can see that relative cardinalities were only defined for features `Gear` and `Web`. In the other cases, the expected behavior is the same of local cardinalities. Also, in the case of `Gear` feature, the cardinality relative to its direct parent `Cartridge` is not defined and should be inferred from the other cardinalities.

To support this requirement the designed language requires to specify only one cardinality for each feature (local or relative) and can automatically infer the remaining relative cardinalities. The inference is achieved by modeling the cardinalities as a constraint satisfaction problem (CSP), according to the consistency constraints described in Section 3.2, and finding a solution that maximizes cardinality ranges.

Figure 6 (a) shows an example feature model where cardinalities in bold face were inferred from declared cardinalities. Because the inference process relies on the cardinality consistency constraints, the inferred cardinalities will always be consistent. In addition, if described cardinalities are not consistent the generated constraint problem will not have a solution. Therefore, the cardinality consistency check and inference are executed as a sole process.

4.3 Automated analysis of feature models

To automatically verify if a configuration conforms to a feature model, we also translate it into a constraint satisfaction problem and use the Choco CSP Solver [15]. We rely on the translation method described in [18], extending it to deal with relative cardinalities. The referenced method considers cardinalities to apply locally, therefore enabling configurations that consider not only the number of feature instances, but also how they are hierarchically organized. For each possible feature instance, a boolean variable is created to represent if the instance is part of a configuration and an integer variable is created to represent the number of instances of the feature type in relation to its parent. For the feature model in Figure 6, for each instance of feature B, we create a boolean variable B_i and two integer variables B_iC and B_iD with domains $[0..10]$ and $[0..2]$ that represent the number of instances of C and D for the i-th instance of B. Additional constraints are added to enforce the minimum number of instances for mandatory features, hierarchical dependencies and group feature cardinalities.

For the semantics of relative cardinalities, besides creating variables representing the number of instances for each direct subfeature, we also create variables that represent the number of instances of indirect subfeatures. Still in the example of Figure 6, it means that in addition to the variables B_iC and B_iD, we create a variable B_iE representing the number of instances of E for this given instance of B. Constraints are added to guarantee that B_iE matches with the number of instances of E for each of its D instances. In the given example, we add the constraint $B_iE = B_iD_1E + B_iD_2E$. The variables generated by this process for the feature model in our example are shown in Figure 6 (b). The variable names are

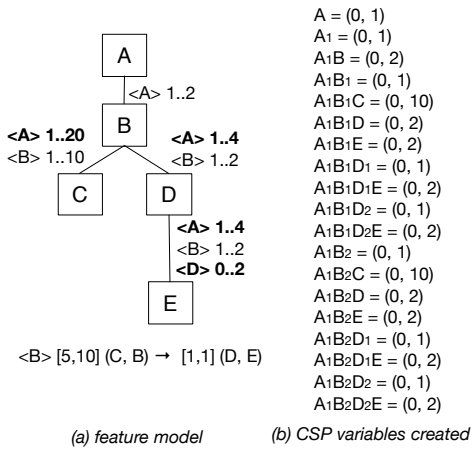


Figure 6: Feature model evaluation.

prepended by A_1 to express that they represent instances that are under the singleton instance of the root feature A.

To optimize the translation and solving of the constraint problem, variables and constraints for relative cardinalities are only added when they are explicitly declared and do not match the implicit cardinality that is derived from the individual local cardinalities. After generating the variables, they can be used for implementing the additional cross-tree constraints using simple logic implication constraints. For example, the constraint <B> $[5..10]$ (C, B) in Figure 6 (a) would generate the following CSP constraint for each i-th instance of B:

IF (B_iC IN $[5..10]$) THEN (B_iD_1E IN $[1..1]$ AND B_iD_2E IN $[1..1]$).

The tools that we developed allow to define feature models with relative cardinalities and configurations of these feature models using a domain specific language. These can then be translated to a constraint satisfaction problem to verify the validity of a configuration or to generate a full configuration from a partial one.

5. EVALUATION

To assess the use of relative cardinalities for feature modeling, we analyze their utility in capturing variability in cloud providers as well as the performance and scalability of the automation solutions proposed in the previous section. Besides this, we discuss the limitations of the work and the threats to its validity.

5.1 Usefulness

To evaluate the usefulness of relative cardinalities we captured variability identified from a set of popular cloud providers into feature models with relative cardinalities. Information from cloud providers was obtained from their documentation and through the use of their configuration tools. Table 1 shows the total number of features and constraining expressions for each modeled provider, as well as how many of them employ local and relative cardinalities.

A feature is considered to have a cardinality if its maximum local cardinality is greater than 1. Features are considered to have a relative cardinality when their cardinalities relative to an ascendant feature do not match the implicit cardinalities derived from intermediary local cardinalities.

A constraint can be composed of many constraining expressions as described in Section 3.3. A constraining expression is considered to employ cardinalities if it constrains a local cardinality whose maximum bound is greater than 1. It is considered to employ relative cardinalities if it applies to a pair of features that are not directly related in the feature model hierarchy.

	Features			Constraining Expressions		
Cloud	Total	card	relCard	Total	card	relCard
Google	69	6	3	6	1	5
Heroku	45	3	1	0	0	0
Jelastic	37	4	5	18	5	13
OpenShift	30	3	2	16	1	9

Table 1: Analysis of cloud providers

As seen from the table, in the studied examples, relative cardinalities describe many relationships and constraints that are not covered by previous feature modeling constructs.

5.2 Scalability

Adding new constructs to feature modeling may bring additional costs to automated feature model analysis, and eventually render it unfeasible. To identify the effects of relative cardinalities in the performance of feature model analysis, we conducted three experiments with feature models of different sizes. First, we evaluated the performance of checking cardinality consistency. Then we verified the performance of translating feature models to constraint satisfaction problems. Finally, we verified the time for checking if a configuration complies to a feature model.

5.2.1 Experimental setup

To execute the experiments, we randomly generated feature models of different sizes (50, 100, 250, 500, 1000, 2500 features). Feature models were generated to be similar to those found in cloud environments and are based on our experience designing the feature models shown in Table 1. Therefore, features in the first three levels of the hierarchy had a 50% chance of having a local cardinality, while in lower levels this chance would be of 1%. For the features chosen to have cardinalities, a random cardinality was generated with the maximum upper bound of 10 instances. Later, for each generated feature model, we randomly selected half of the features for which local cardinalities were defined and added extra relative cardinalities to them.

For each feature model size, 50 different models were generated and had relative cardinalities added. From this process, we obtained two sets of feature models, one composed of random feature models with local cardinalities, and another composed of the same models augmented with relative cardinalities.

Using the generated feature models we performed experiments to evaluate the scalability of a) the consistency check and inference mechanism; b) the translation to constraint satisfaction problem; and c) the validation of a configuration.

All experiments were run on a MacBook Pro Computer with a 2 GHz Intel Core i7 processor and 8GB of memory.

5.2.2 Results

For the first experiment we measured the time to verify the consistency between cardinalities and infer cardinalities that were not described in the model. This includes the time to translate the consistency conditions described in Section 3.2 into a constraint satisfaction problem and the time to solve the problem.

This experiment was ran only for the set of feature models with relative cardinalities. The average measured time, by number of features of the model, is shown in Table 2.

For the second experiment, for both sets of feature models we

Features	Average time (ms)	Standard deviation (ms)
50	63.80	17.60
100	220.16	60.79
250	796.82	194.29
500	3,627.55	698.80
1,000	14,923.11	4,438.63
2,500	129,472.05	38,772.75

Table 2: Average time for consistency checking and cardinality inference.

measured the time to fully translate a model to a constraint satisfaction problem. Figure 7 shows, in a logarithmic scale, the distribution of execution times by the number of features for both sets of feature models.

As many random models were generated for each number of features, we obtained many different translation times for the same number of features. This happens because according to how the hierarchy was generated and where the cardinalities were placed, the number of instances that can be generated by the feature model may vary greatly. Still, as we can see from the chart, translating a feature model with relative cardinalities introduces an overhead into the processing, which in the conducted experiments was on average of 41%.

Despite the overhead added by relative cardinalities in the translation, it is still feasible to use them, and in the worst case found in our experiments it took less than 10 minutes to translate a feature model with 2500 features that can generate configurations with up to 187,770 feature instances. Feature models from cloud providers have usually far less features, but can still have a large number of possible feature instances.

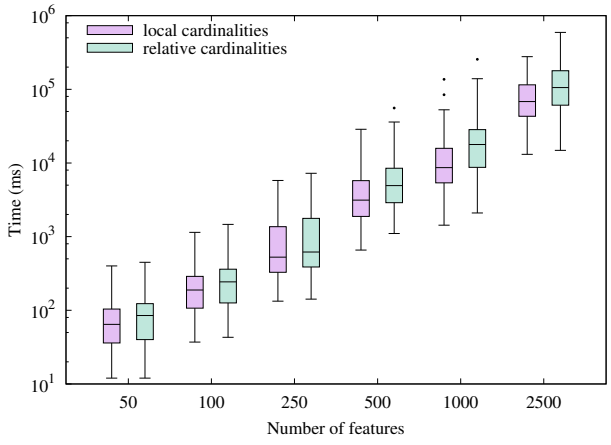


Figure 7: Distribution of time to translate feature model to constraint satisfaction problem

For the last experiment we evaluate the time to check if a configuration conforms to a feature model. Using the translations obtained in the previous experiment we generate valid configurations for feature models in both sets and check their compliance to their respective feature model. Figure 8 shows the time distribution for checking a configuration for feature models with local and relative cardinalities. This time includes the translation of the configuration, together with the conversion of feature model constraints to a constraint satisfaction problem, and the resolution of the problem. From the graph we can see that introducing relative cardinalities did not affect the time to validate a configuration. This is linked to the fact that configurations for feature models with relative cardinalities tend to be smaller as relative cardinalities usually restrict the number of allowed instances of a feature.

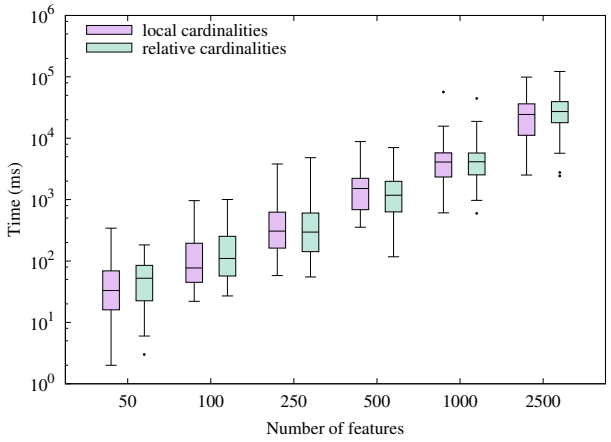


Figure 8: Time to verify compliance of a configuration to a feature model

5.3 Threats to validity

Though we have studied and worked extensively with cloud provider configurations to propose relative cardinalities, we only formally modeled a small number of them for extracting data. Still, other domains such as cyber-physical systems [24] exhibit similar complex variability and could benefit from this extension.

The feature models generated to measure the scalability follow a pattern where cardinalities are mostly concentrated in the upper levels. As the hierarchy of a feature model and placement of cardinalities may greatly influence the size of the translated constraint satisfaction problem, it can also influence the performance of feature model processing negatively. We tried to ensure that the solution is feasible for models similar to those we identified for the cloud domain.

6. RELATED WORK

Cardinalities were originally introduced to feature models [23] as an application of UML multiplicities to feature groups from the original FODA notation [16]. Feature cardinalities were first introduced in [4], motivated by practical applications. *Czarnecki et al.* [5] integrated feature model extensions, such as attributes, group and feature cardinalities to formalize the concept of cardinality-based feature models [6]. Our approach relies on this concept and the preceding developments in the field of feature models.

In [7], the authors discuss the need for additional constraints that deal with cardinality-based feature models, and for constraints that apply only in a given scope of the feature model. However, the proposed constraints are defined in OCL, which is a general purpose constraint language and whose evaluation may limit the performance of automated analysis of feature models. Similarly, XPath has also been employed as a notation for describing additional constraints in feature models [2].

In [19], *Michel et al.* discuss about global and local interpretations of feature cardinalities and argue for choosing the local interpretation for feature cardinalities. The authors also discuss about

the need to update additional constraints to consider feature instances, but do not present any proposal to deal with it.

Quinton et al. [21] proposes a constraining language that extends *requires* constraint with cardinalities and allows for constrains to apply in local or global scope. In [17], authors propose a feature modeling language where cross-tree constraints can be applied to individual instances. Though it potentially allows for expressing any kind of constraint between instances, describing constraints individually for each possible instance is not feasible when we have thousands of possible feature instances.

CardyGAn [26] is a tool for analysis of consistency in cardinality feature models with unbounded cardinalities. Like in our approach, it provides a domain-specific language for describing feature models and configurations and can be used for the generation and validation of configurations.

Feature modeling has been employed to capture variability in cloud environments, but usually in a constrained context. In [8, 10] regular feature models are used to model variability within virtual machine configurations, while in [22] cardinality-based feature models capture variability in cloud providers.

In summary, what distinguishes our proposal from previous work is that we generalize the interpretation of cardinality scope. Instead of choosing for a global or local interpretation we enable users to choose on a per case basis the most appropriate interpretation. Concerning cloud environment modeling, our motivation for using relative cardinalities is to enable capturing all variability in cloud providers, within and across different contexts such as applications, virtual machines, projects, etc.

7. CONCLUSION

In this paper we introduced the concept of relative cardinalities for feature models. Based on our investigation on managing variability in cloud computing platforms, we identified limitations found in feature cardinalities and the need to extend their interpretation. We proposed a definition of cardinality-based feature models with relative cardinalities, analyzed the issue of consistency between cardinalities, and updated cross-tree constraints to take into account relative cardinalities.

In addition, we proposed a metamodel and editing tools for describing feature models with relative cardinalities, as well as a translation process into constraint satisfaction problems for automatic processing. We also demonstrate that relative cardinalities are valuable for modeling variability in cloud computing configurations and that automated reasoning upon them is feasible. We have fully implemented and thoroughly tested our approach. The implementation and research results can be found in the accompanying site[2].

As future work, we plan to integrate feature models from multiple cloud providers into a into a multi-product line approach [1, 12]. Our goal is to support the automatic configuration of multi-cloud environments for service-based applications. This includes taking into account requirements such as scalability, redundancy and location, which will lead to constraints across multiple cloud providers and their respective feature models. Due to the huge number of possible multi-cloud configurations, we intend to evaluate the use of search-based strategies [11] combined with constraints [13] for reasoning on multiple product lines. Another research direction is to extend variability management to runtime [14] in order to support self-adaptive multi-cloud configurations.

[2]http://researchers.lille.inria.fr/~sousa/relativecard/

8. REFERENCES

[1] M. Acher, P. Collet, A. Gaignard, P. Lahire, J. Montagnat, and R. France. Composing multiple variability artifacts to assemble coherent workflows. *Software Quality Journal*, 20(3-4):689–734, 2012.

[2] M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature modeling plug-in for eclipse. In *Proceedings of 2004 OOPSLA Workshop on Eclipse Technology eXchange*, pages 67–72, New York, NY, USA, 2004. ACM.

[3] L. Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.

[4] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. Generative programming for embedded software: An industrial experience report. In D. Batory, C. Consel, and W. Taha, editors, *Generative Programming and Component Engineering*, volume 2487 of *Lecture Notes in Computer Science*, pages 156–172. Springer Berlin Heidelberg, 2002.

[5] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In R. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 266–283. Springer Berlin Heidelberg, 2004.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[7] K. Czarnecki and C. H. P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*, pages 16–20, 2005.

[8] A. Ferreira Leite, V. Alves, G. Nunes Rodrigues, C. Tadonki, C. Eisenbeis, and A. Magalhaes Alves de Melo. Automating resource selection and configuration in inter-clouds through a software product line method. In *Proc. IEEE International Conference on Cloud Computing, (CLOUD'15)*, pages 726–733, New York, United States, June 2015.

[9] N. Gamez and L. Fuentes. Software product line evolution with cardinality-based feature models. In K. Schmid, editor, *Top Productivity through Software Reuse*, volume 6727 of *Lecture Notes in Computer Science*, pages 102–118. Springer Berlin Heidelberg, 2011.

[10] J. García-Galán, P. Trinidad, O. F. Rana, and A. Ruiz-Cortés. Automated configuration support for infrastructure migration to the cloud. *Future Generation Computer Systems*, 55:200 – 212, 2016.

[11] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. Search based software engineering for software product line engineering: A survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 5–18, New York, NY, USA, 2014. ACM.

[12] H. Hartmann, T. Trew, and A. Matsinger. Supplier independent feature modelling. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 191–200, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[13] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE 2015)*, 2015.

[14] M. Hinchey, S. Park, and K. Schmid. Building dynamic software product lines. *Computer*, 45(10):22–26, 2012.

[15] N. Jussien, G. Rochart, and X. Lorca. Choco: an open source Java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.

[16] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[17] A. S. Karataş, H. Oğuztüzün, and A. Doğru. From extended feature models to constraint logic programming. *Science of Computer Programming*, 78(12):2295 – 2312, 2013. Special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of {FSEN} 2011).

[18] R. Mazo, C. Salinesi, D. Diaz, and A. Lora-Michiels. Transforming Attribute and Clone-Enabled Feature Models Into Constraint Programs Over Finite Domains. In *6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, Beijing, China, June 2011.

[19] R. Michel, A. Classen, A. Hubaux, and Q. Boucher. A formal semantics for feature cardinalities in feature diagrams. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 82–89, New York, NY, USA, 2011. ACM.

[20] C. Quinton, A. Pleuss, D. L. Berre, L. Duchien, and G. Botterweck. Consistency checking for the evolution of cardinality-based feature models. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 122–131, New York, NY, USA, 2014. ACM.

[21] C. Quinton, D. Romero, and L. Duchien. Cardinality-based feature models with constraints: A pragmatic approach. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 162–166, New York, NY, USA, 2013. ACM.

[22] C. Quinton, D. Romero, and L. Duchien. SALOON: a platform for selecting and configuring cloud environments. *Software: Practice and Experience*, 46(1):55–78, 2016.

[23] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Extending feature diagrams with uml multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, volume 23, 2002.

[24] D. Romero, C. Quinton, L. Duchien, L. Seinturier, and C. Valdez. *Software Architecture: 9th European Conference, ECSA 2015, Dubrovnik/Cavtat, Croatia, September 7-11, 2015. Proceedings*, chapter SmartyCo: Managing Cyber-Physical Systems for Smart Environments, pages 294–302. Springer International Publishing, Cham, 2015.

[25] A. Safilian, T. Maibaum, and Z. Diskin. The semantics of cardinality-based feature models via formal languages. In *FM 2015: Formal Methods*, pages 453–469. Springer, 2015.

[26] T. Schnabel, M. Weckesser, R. Kluge, M. Lochau, and A. Schürr. Cardygan: Tool support for cardinality-based feature models. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '16, pages 33–40, New York, NY, USA, 2016. ACM.

[27] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[28] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A bdd-based approach to verifying clone-enabled feature models: Constraints and customization. In H. Mei, editor, *High Confidence Software Reuse in Large Systems*, volume 5030 of *Lecture Notes in Computer Science*, pages 186–199. Springer Berlin Heidelberg, 2008.